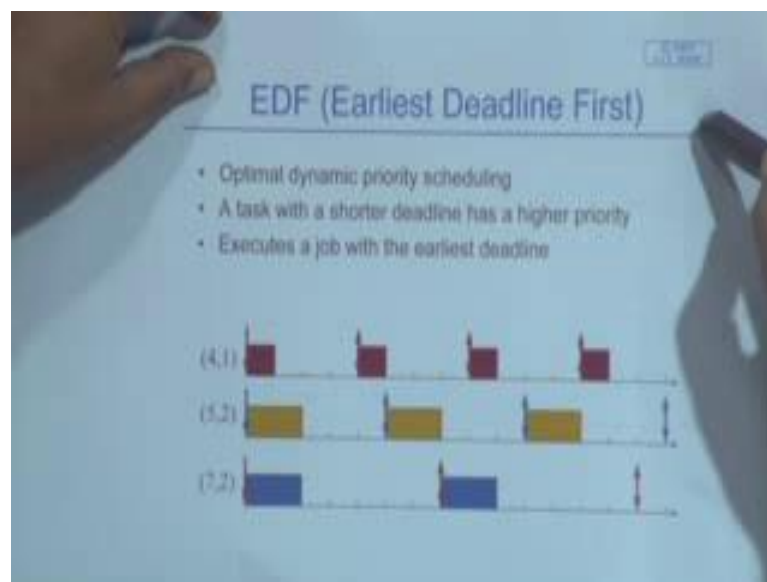


Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 29
EDF Algorithm and Resource Constraint Issue

In the last lecture we are looking at the rate monotonic scheduling which was a static priority scheduling, what did we mean by that? There the priority is determined by the period of the task and since the period is known apriori that is also fixed and we have seen that under certain condition that is guaranteed RMS is guaranteed to give a schedule.

(Refer Slide Time: 01:00)

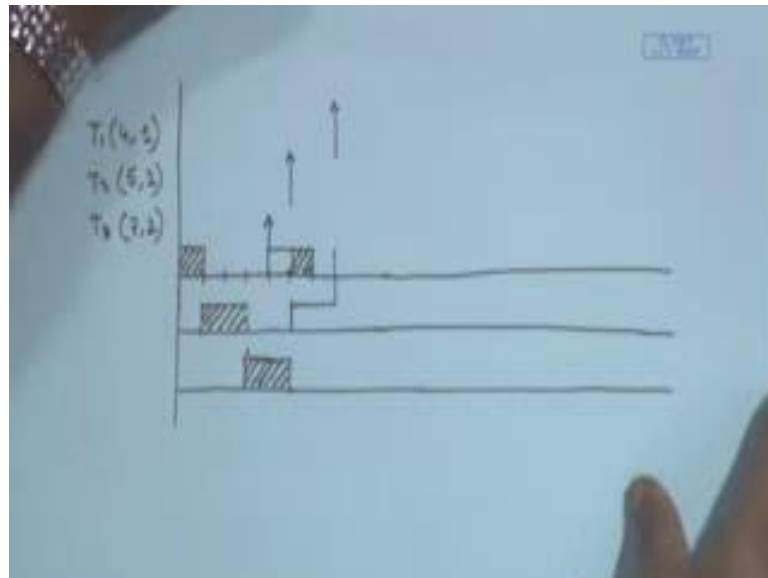


Now, here we are now introducing a dynamic priority scheduling; why are we calling it dynamic? Because the priority is computed every time a task is to be allocated, I have got the tasks T 1, T 2, T 3 once one task completes then from those bank of other tasks have to select another task. I have to select another and at that time I am re-computing the priority based on the deadline that is at your disposal, the shorter the deadline, the higher the priority; at that point. Suppose something added deadline of 7 and because of allocation of the earlier tasks 3 units have gone then your deadline remaining is 4.

So, this earliest deadline first actually executes a job with the earliest deadline. Now we start with the same set of jobs that we had taken as an example for the rate monotonic

scheduling, same execution times and same periods and I hope that you have got this noted down. Now we will try to and you if you recall with this job makes, with this set of jobs we had faced a problem that the third job missed its deadline. So, let us see what happens this time, so with these job; let us do the scheduling first; so again.

(Refer Slide Time: 03:04)



Let me make the times, so 1, 2, 3, 4, so that is the period for the first task another one is 5. So, this is for the next one, second one is here and the third one is 7, so 5, 6, 7. So, that is somewhere here there is a third one. Now right now whose deadline is earliest task T 1s deadline is earliest, so initially task T 1s deadline is earliest, so I write down again. So, T 1 will now be executed and T 1 will be executed for, so here are my tasks T 1 which is 4 1; T 2 which is 5 2; T 3 which is 7, 2. So, T 1 will be executed for 1 unit and T 1 has finished I will follow this time chart for T 1 then here will be 42 and here will be 43 that will more convenient for me and also for you to visualize.

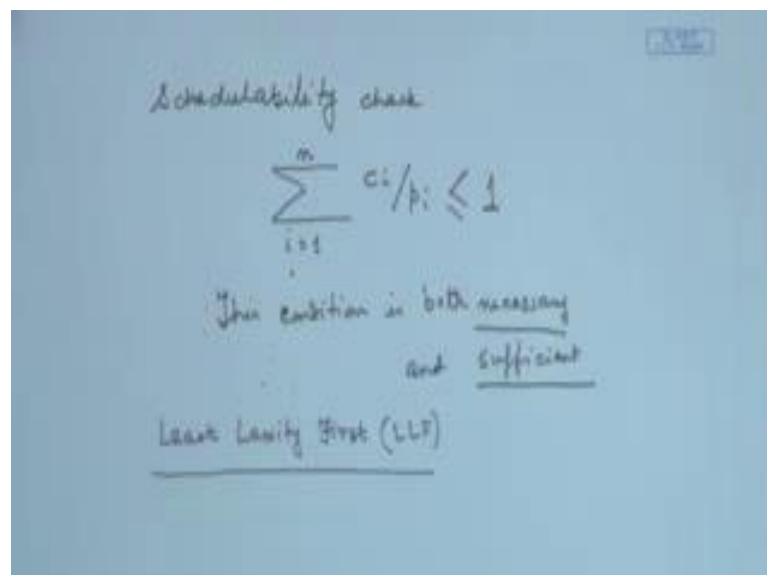
Now, the second task is this one, so it has got to start from this point onwards because the deadline; now what is the deadline of this? What is its current deadline? 4 its current deadline is 4 because I start from this point as current deadline is 4 and its deadline is 6. Therefore, I select this and take this for 2 units it executes for 2 units then, so 2 and 1 gone, so 3 gone. So, its deadline is next this one will be allocated right which one will be allocated next this one, why the first one? First one has still not come. So, this one has to this one will start; now at this point it has gone up to this, at that point the first one has

come; at this it required this 1 required 2 units, it has executed 1 unit and by that time the first one has come. So, will the first one preempted; I will have to again now compute since the task has arrived, let us compute the deadline.

Its deadline is 4; 4 at this point and its deadline is how much no no no its deadline was starting from here, it arrived here actually. So, its deadline is 3 because 1 has gone, so this one will have the priority, so this will complete. You see the difference between what happened with rate monotonic scheduling, so this could be completed at this point. Now there are 2 tasks has the other one arrived, this one has arrived now at this point, the second task has arrived demanding 2 units of time; out of this which one will continue; why T 1 at this point T 1s is 3 units because it had 4 and 1 is a it has waited for. So, 3 units left for deadline and this one has just arrived and has got 5 units. So, this one will continue and this one will continue from here, in this way we carry out the earliest deadline first algorithm is it clear?

Now, in this way we carry out the earliest deadline first and we also have again some conditions for this when I can schedule it and when I cannot schedule it. So, the schedulability check that we had for RMS.

(Refer Slide Time: 08:29)



Schedulability check

$$\sum_{i=1}^n c_i/p_i \leq 1$$

This condition is both necessary and sufficient

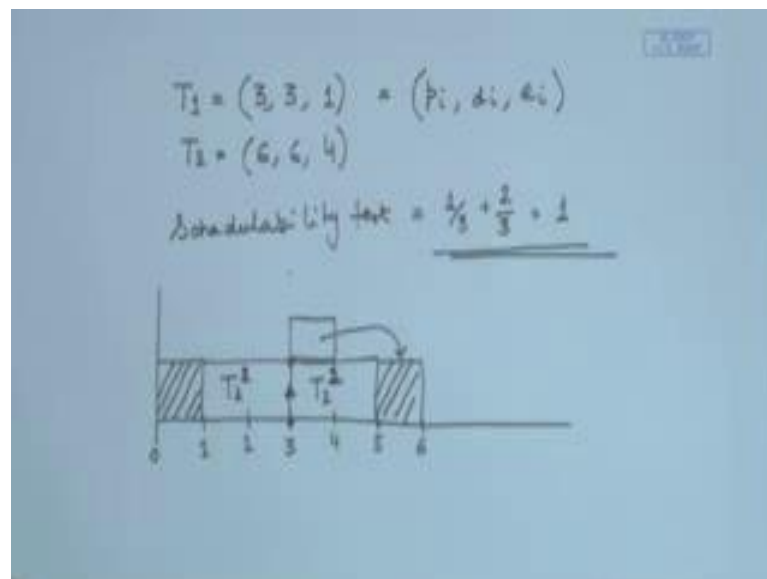
Least Laxity First (LLF)

Similarly, for this here we have got the schedulability check which can be done offline for n number of tasks. Remember this schedulability check is for a uniprocessor; obviously, I do not have the luxury of having multiple processor where I can schedule

multiple tasks then the scenario would have been different, here we are talking of a uniprocessor environment and the schedulability condition is 1 to n and this condition is both necessary and sufficient a stronger condition and there is another algorithm called the least laxity first or LLF; which has got the same schedulability check.

Now, the LLF has the name implies you can easily find out how similar to the deadline, what will you do in that case; at every point you look at the laxity, what is the computation time left? What is the computation time that is left and that at every time for example, let us take this scenario a job say for example, this has completed straightaway. Suppose a job as this one, I had to take a decision at this point right after this job completed 1 unit because this other job arrived. So, here I sit back and look at the laxity; laxity means from this time, how much computation time is left? 1 unit and how much is left from the deadline, so from there I look at the laxity and whichever has got the least laxity I will schedule that first. So, for that the same schedulability criteria applies.

(Refer Slide Time: 11:32)



Now, let us then workout a task set and do it; suppose I have got a task set T 1. Now here I am using a little different notation 3, 3, 1 means it is basically I am giving the priority sorry the period I am always whenever I need to say period I even talk a priority and that is true for rate monotonic. Now deadline and execution time or c_i , e_i or c_i whatever you write. So, here I am saying that the deadline need not be the same as the period and

that is the case for earliest deadline first, the period and the deadline need not be the same although in the example that I took; I had taken both to be the same.

Now, suppose there is another task T 2 which is 6, 6, 4 here again I am showing that they are same, but now if I have those I can carry out the schedulability test let me carry out the schedulability test what will that be will be again $\sum c_i/p_i$, so 1 by 3 plus 2 by 3, so less than equal to 1 right; therefore, it should be schedulable for sure.

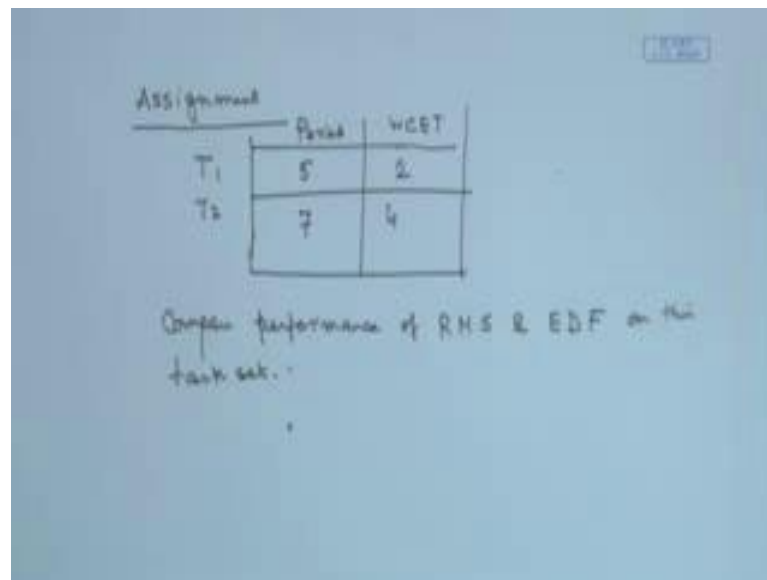
So, let us try to schedule this, so I am taking this to be 0, this to be 1 then 2, 3. Now first which one by earliest deadline first we can see the deadline of this is earlier 3. So, task T 1 will be scheduled and you will take 1 unit; here I am assuming that all of them have arrived at the same time. So, given that both of them are available at this point I am trying to do the schedule, the second job whose priority is coming next will require 4 units of time, but its period is 3 for this. So, it will again come here and this one will require 4 units of time, so 1 and 2 units it executes.

So, this is T 2 first run of T 2; 1 this is the first run of T 2 then at this point again T 1 has arrived because T1s period is 3. So, T 1 has arrived here and is waiting to be executed, so at this point I look at the deadlines of this.

The deadline of T 2 was it has executed for 2 units and 1 unit it had waited 3 units. So, it is 3; its deadline is 3 and its deadline is also 3, so you can take any one of them. So, upto 3 it works then again I continue T 2; T 2 continues and T 2 will come again for what was T 2s execution time 4 units. So, T 2 more 1, 2 here it completes T 2; T 2 second run completes these are not the second run actually the first run continuing again. So, T 2 1 is continuing and then this one will take 1 unit. So, it can come and be done here T 1 can come although it arrived here it comes here. So, in that case I can schedule both the jobs as this being shown that this condition is being met. So, that is the earliest deadline first algorithm.

Now, I leave to you to look at take down another assignment.

(Refer Slide Time: 16:43)



The image shows a handwritten table on a blue background. The table is titled 'Assignment' and has two columns: 'Period' and 'WCET'. There are two rows of data. The first row is for process T₁ with a period of 5 and a WCET of 2. The second row is for process T₂ with a period of 7 and a WCET of 4. Below the table, there is handwritten text that reads 'Compare performance of RMS & EDF on this task set.'.

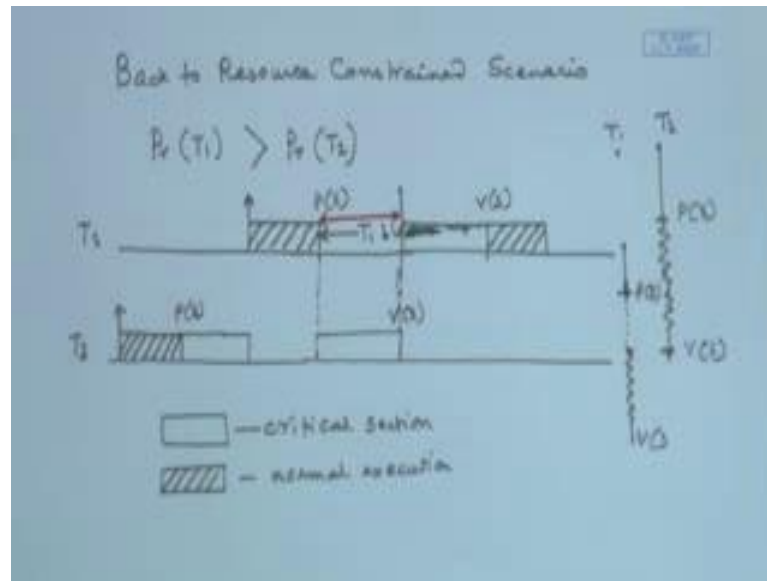
Assignment	Period	WCET
T ₁	5	2
T ₂	7	4

Compare performance of RMS & EDF on this task set.

There are processes T₁ and T₂ and the period and worst case execution time are given as T₁ is 5, 2, 7, 4 I want you to carry out a comparison, the performance of RMS and EDF on this task set and show which one performs better.

So, next will move to the resource constraint scenario but till now let me quickly summarize what we have done. We have looked at two very important scheduling algorithm that are used for real time systems, real time operating systems and they are rate monotonic scheduling and earliest deadline first one is static scheduling static priority other one is a dynamic priority and we have seen that there is a schedulability check condition for both of them and we can test whether the tasks are schedulable or not for RMS, the condition was sufficient, but not necessary; that means, even if the schedulability test fails that does not guarantee that no schedule exists still there can be a schedule possible, while the schedulability test for EDF or earliest deadline first is both necessary and sufficient. Next, we will move to again the scenario with resource constraints which we were discussing in the last class.

(Refer Slide Time: 19:34)



So, we had seen the problem of having shared resource, now so again back to resource constraint scenario. Till now this EDF and RMS that we have discussed that does not consider any scarcity of resource, but as we have seen in the earlier lecture that there are shared resources where we access the shared resources using semaphores which ensure mutual exclusion. Now suppose I have got again 2 tasks priority of task T 1 is greater than priority of task T 2, the priority can be the deadline priority can be anything.

Now, suppose now the arrival time of the task may be different, so here I am considering task T 1 and task T 2 where there will be conflict over resources. Now suppose task T 2 has arrived first and has started, task T 2 is executing this is a normal execution; that means, a task T 2 is executing and no request for any shared resource has been made, but suppose at this point it asks for a shared resource; that means, it wants to enter the critical region here. So, at this point it makes the request for the shared resource P S and it continues; is continuing this non hashed part is the critical section execution. So, whenever I am giving this sort of thing this is critical section whereas, this hashed part is normal execution.

So, now this was doing critical section here; that means, somewhere here it has got in the critical section, but as not yet completed it has not yet come out. Now at that point of time task T 1 comes now; obviously, it is a preemptive scenario. So, task T 1 will get

priority because I have already said that because of some reason, the priority of T 1 is greater than priority of T 2.

So, suppose now at this point T 1 starts working and T 1 starts working in the normal mode, but look at T 1 here T 1 started much later T 1 started from this point and here T 1 makes a request to enter the critical section, it makes a request P S right at this point; that means, it also wants to enter the critical section, but since this 1 is already in the critical section this one cannot get in. So, now this is getting blocked; T 1 is getting blocked here how long will T 1 get blocked?

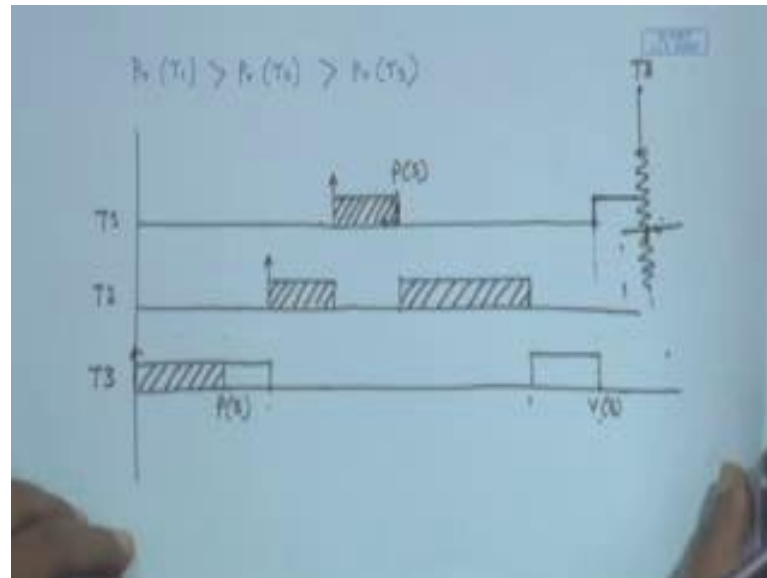
Now, at this point you want gets blocked therefore, now T 2 can continue and complete the critical section. So, suppose this is doing V S here all right; that means, it is doing V S here, now the P S of this had done a P S here, now the P S can succeed. So, actually it will not be blocked up to this at this point it will start its critical section and then. So, at this point it will its waiting, waiting, waiting and here it enters the critical section and completes it and does a V S and after doing the V S here, it can continue its normal execution and complete that is revisit it once again. So, this blocking part is only upto this let me put a color here red color, I do not know whether this part is the blocking part this is the part for which it is blocked.

So can you tell me what is the maximum duration for which in two process scenario it can be blocked? Is a length of the critical section, it may be that both of them it just requested immediately after the other one entered the critical section. So, the length of the critical section of the code, the execution time of that which is the maximum duration for which it can be blocked. Therefore, you see because of earlier we had that worst case execution time known to us.

So, given that worst case execution time, we computed the schedule for in both RMS and EDF, but here you see the worst case execution time is being affected by the availability of the resources and whether other processes are capturing those resources. So, that is how here because of mutual exclusion, we can get a blocking and the execution time can be delayed. So what can be done in such scenarios that is the next level of challenge for getting the schedules.

So, let us look at another scenario where blocking of here I have said that the blocking of a process can be maximum up to the length of the critical section, but if there be more than two processes then the scenario may be a little worse.

(Refer Slide Time: 27:26)



Let us see how; let us take a 3 tasks scenario priority of T 1 is higher than the priority of T 2 and that is higher than the priority of T 3. So, T 3 has got the lowest priority; that means, what T 2 can preempt T 3 and T 1 can, so T 2 can preempt T 3 and T 2 can prevent T 3 from releasing a resource you understand what that means.

Suppose T 3 is executing and it has captured some resource and now T 2 comes T 2 preempts it because T 2 has got the highest priority, as T 2 preempts it this one has not released the resource and T 2 will not need that resource. Therefore, the T 2 will have to again will be waiting write. So, let us look at the scenario again in the same way and let us see how blocking takes place here. So, let us assume T 3 is starting here T 3 is starting here T 3 has done some normal execution and has executed a P S; that means, it is taking a critical section and has gone up to this not done V S; you see it has done P S means it has grabbed the resource V S means it has released the resource. So, the resource at not at been released T 2 has come here.

T 2 has come here, so T 2 will preempt T 3 will have to leave the resource for T 2. So, now, this one is working which one? T 3 was in the critical section was in it has not yet requested for any critical section, its normal execution hashed 1 is normal. So, this is in

the critical section, but it had to leave the CPU. So, the question is this is in the critical section, so what is happening here? Is this suppose this is T 3, T 3 was going on doing something, at this point it has asked for shared resource might be some file and it needs the file up to this, up to this time may be.

Now at this point it has been preempted by T 2 higher priority task. So, it had to leave the CPU, it had has left the CPU, so it cannot do the file handling.

So, it is actually waiting here and at this point, suppose it was doing something T 2 was executing and T 2 has not requested for any critical section. Now T 1 only sudden appears here; now so T 1 has got the highest priority and so T 1 will start executing and now T 1 does a P S, it also wants to have that thing. Now what will happen? At this case will T 1 get the resource? No it will not get the resource.

So, then T 1 goes in the wait state its waiting and then who will get the resource, now CPU T 2 will get the resource now. So, now at this point T 2 will now start from here and will continue and this poor fellow T 3 being of the lowest priority is waiting and this fellow is also waiting because a poorer guy has caught hold of that; poorer in the sense lower priority guy, lower priority person has got is lower priority process is holding the resource. So, this rich man who wanted to have favor is not getting that either right and in between this middle man has done it.

Now, after this T 2 completes this what will happen then who will get to run? T 3 because T 3 is holding the resource, this T 2 has got nothing to do. So, now this one will do a little more of file handling and do V S here.

Student: What (Refer Time: 33:15).

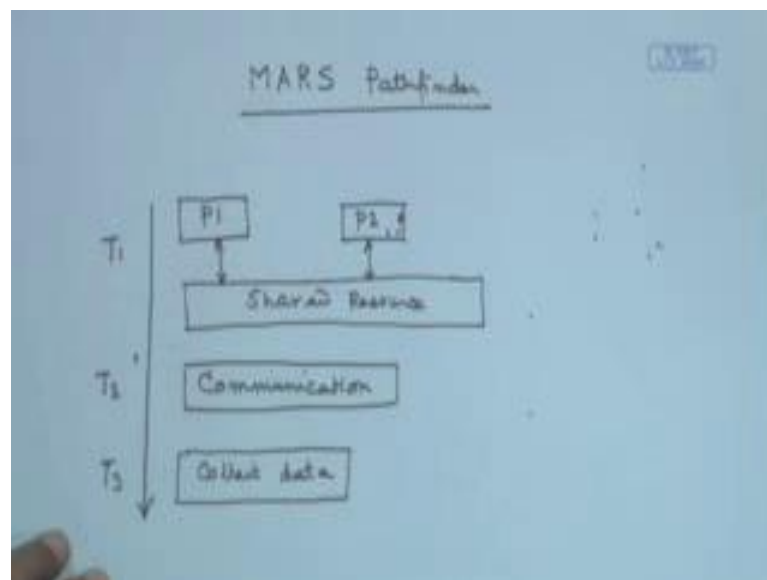
Then again T 2 would get it if T 2 had asked for P S the same this was then T 3 will come will show that what will happen. So, what is happening here? Now it does V S here; now this person, now can get hold of the T 1 can start from here; T 1 can resume from here. So, what is the blocking time of T 1? Too much, because this entire period has been blocked last this period, so it is a beyond and this one has been blocked much more than the length of the critical section. So, with more than two processes, the blocking can exceed the length of the critical section.

So, now let us tell a story.

Student: T 2 what is the (Refer Time: 34:33) that time. So, (Refer Time: 34:34) given to T 2 (Refer Time: 34:35) T 1.

The question is if T 2 had asked for a the critical section somewhere here say here all right, then whether if it asked for P S then who is holding that? Now that this will also go in wait then T 3 will come T 3 will execute. Once you have done the P S unless you do the V S no one else can proceed. So, after this one; now when it comes back then who will get out of these 2 T 1 will get because T 1 is of higher priority and also wait.

(Refer Slide Time: 35:26)



Now, there was the mission; the mars mission some of you might know about that. There is a mars path finder, the pathfinder went on mars and it was supposed to send some information from time to time to the earth. So, it had 3 tasks it was there were some com I mean the system had number of processors and processors which were communicating among them through some shared resource and that was a very frequent task; they were getting the shared resource.

Now, that was a very frequent task that was of the highest priority let us call it T 1 the lowest priority task was that is lowest priority in the sense that was rather infrequent is collect meteorological data; that was done at a lesser frequency that is another job and there was in between, there was another task of communication might be the

communication with the earth and whatever that is the communication job. So, these were the 3 priorities tasks were in this order of priority. It was found that often there is being a system reset and the entire thing whatever data was being collected is being lost and is not being sent to the earth.

So, what was the problem for that the problem I mean press reported that the system is trying to do too many things at a time etcetera but ultimately what was found out is scenario like this that there was a priority inversion what was happening there. Let us see a scenario and what was happening is actually shown in this sort of scenario see here this one is being blocked for a long time right why because of holding the critical section, this one is getting the priority back and it can hold on it.

If I had given, so consequently what is happening this one is being delayed for a longer time. Now as there are main process was the shared memory access process was being delayed for a long time, the system was there was a watchdog timer in the system and you know what a watchdog timer is; it looks for some healthy situation signals, it was not getting that signal for a long time and it was assuming that there is something wrong and it applied system reset.

Now, it was later on solved by the priority inversion approach, what was happening is the priority was being inverted and that was being solved by another protocol called the priority inheritance protocol. So, I think I will be discussing that in the next class and then we will proceed further.